Document: GWD-Perf-16-2
Category: Informational
Obsoletes: GWD-Perf-16-1
Expires: December, 2002

Ruth Aydt
University of Illinois, Urbana-Champaign
Dan Gunter
Lawrence Berkeley National Laboratory
Warren Smith
NASA
Martin Swany
University of Tennessee, Knoxville
Valerie Taylor
North Western University
Brian Tierney
Lawrence Berkeley National Laboratory
Rich Wolski
University of Tennessee, Knoxville
December, 2001

A Grid Monitoring Architecture

Abstract

Large distributed systems such as Computational and Data Grids require a substantial amount of monitoring data be collected for a variety of tasks such as fault detection, performance analysis, performance tuning, performance prediction, and scheduling. Some tools are currently available and others are being developed for collecting and forwarding this data. The goal of this paper is to describe a common grid monitoring architecture with all the major components and their essential interactions. To aid implementation, we also discuss the performance characteristics of a Grid Monitoring system and identify areas that are critical to proper functioning of the system.

1 Introduction

Performance monitoring of distributed computing components is critical for enabling high-performance distributed computing. Monitoring data is needed to determine the source of performance problems and to tune the system and application for better performance. Fault detection and recovery mechanisms need monitoring data to determine if a server is down, and whether to restart the server or redirect service requests elsewhere [10][14]. A performance prediction service might use monitoring data as inputs for a prediction model [16], which would in turn be used by a scheduler to determine which resources to use.

There are several groups that are developing Grid monitoring systems to address this problem [9] [11] [14][16] and these groups have recently seen a need to interoperate. In order to facilitate this, we have developed an architecture specific to the needs of a Grid monitoring system. A Grid monitoring system is differentiated from a general monitoring system in that it must be scalable across wide-area networks, and include a large number of heterogeneous resources. Its naming and security mechanisms must also be integrated with other Grid middleware. We believe the Grid Monitoring Architecture (GMA) described here addresses these concerns and is sufficiently general that it could be adapted for use in distributed environments other than the Grid. For example, it could be used with large compute farms or clusters that require constant monitoring to ensure all nodes are running correctly.

1.1 Design Considerations

With the potential for thousands of resources at geographically different sites and tens-of-thousands of simultaneous Grid users, it is important for the data management and collection facilities to scale while, at the same time, protecting the data from spoiling. In order to allow scalability in both the administration and performance impact of such a system, decisions about what is monitored, measurement frequency, and how the data is made available must be made locally to the monitoring activity. Thus, instead of a centralized monitoring server, there are many independent monitoring components. To bind the system together, these components place metadata describing their state in a central directory service, which may itself be physically distributed. Localizing the monitoring responsibilities also helps minimize the effects of host and network failure, making the system more robust under precisely the kinds of conditions it is trying to detect.

In some models, such as the CORBA Event Service, all communication flows through a central component, which represents a potential bottleneck. In contrast, we propose that performance event data, which makes up the majority of the communication traffic, should travel directly from the producers of the data to the consumers of the data. In this way, individual producer/consumer pairs can do "impedance matching" based on negotiated requirements, and the amount of data flowing through the system can be controlled in a precise and localized fashion based on current load considerations. The design also allows for replication and reduction of event data at intermediate components acting as consumer/producer caches or filters. Use of these intermediate components lessens the load on producers of event data that is of interest to many consumers, with subsequent reductions in the network traffic, as the intermediaries can be placed "near" the data consumers. Because the directory service contains only metadata, with careful design it will not be a bottleneck.

We also considered a purely SNMP-based solution for monitoring, but rejected it because we felt that the SNMP simple GET/SET model is not rich enough, as there is no support for subscription. Also, it is not clear that the security model maps well to the Grid Security Infrastructure. However, we definitely envision the use of SNMP-based tools as a source of monitoring data.

2 Architecture and Terminology

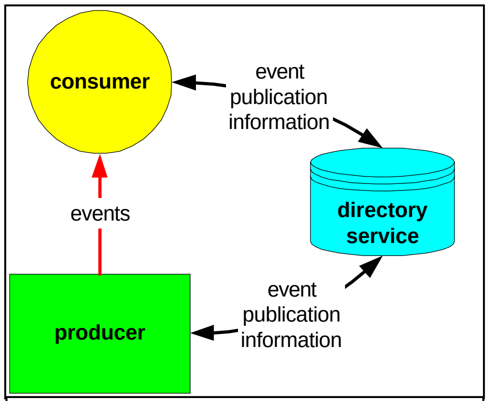

Figure 0: Grid Monitoring Architecture Components

The Grid Monitoring Architecture consists of three types of components, shown in Figure 1:

- Directory Service: supports information publication and discovery
- Consumer: receives performance data (performance event sink)
- Producer: makes performance data available (performance event source)

The GMA is designed to handle performance data transmitted as timestamped (*performance*) *events*. An event is a typed collection of data with a specific structure that is defined by an *event schema*. Performance event data is always sent directly from a producer to a consumer.

The GMA architecture supports both a streaming publish/subscribe model, similar to several existing Event Service systems such as the CORBA Event Service [1], and a query/response model. For both models, producers or consumers that accept connections publish their existence in a directory service. Consumers can use the directory service to discover producers of interest and producers can use the directory service to discover consumers of interest. Either a producer or a consumer may initiate the connection to a discovered component. Communication of control messages and transfer of performance data occurs directly between each consumer/producer pair without further involvement of the directory service.

3 Components and Interfaces

In this section we further define the functionality and interfaces of the consumer, producer, and directory service components.

3.1 Directory Service

To describe and discover performance data on the Grid, a distributed directory service for publishing and searching must be available. The GMA directory service provides information about producers or consumers that accept requests. When producers and consumers publish their existence in the directory service they typically specify the event types they produce or consume. In addition, they may publish static values for some event data elements, further restricting the range of event data that they will produce or consume. This publication information allows other producers and consumers to discover the types of event data that are currently available, the characteristics of that data, and the sources or sinks that will produce or accept each type of data. The directory service is not responsible for the storage of event data -- only per-publication information about which event instances can be provided. The event schema may, optionally, be made available by the directory service.

The functions supported by the directory service are:

1. Authorize search: establish identity of a client for searching.
2. Authorize modify: establish the identity of a client for modifying entries.
3. Add: add a record to the directory.
4. Update: change the state of a record in the directory.
5. Remove: remove a record from the directory.
6. Search: perform a search for a producer or consumer of a particular type and possibly with fixed values for some of the event elements. The client should indicate whether only one result, or more than one result, if available, should be returned. An optional extension would allow the client to get multiple results one element at time using a "get next" query in subsequent searches.

Query-optimized directory services such as LDAP [15], Globus MDS [3], the Legion Information Base, and the Novell NDS, all provide the necessary base functionality for this service, but only in their fully distributed implementations. Some public-domain implementations of these services do not support distributed implementation.

3.2 Producer

A producer is any component that sends events to a consumer. The functions supported by a producer are listed below. The first functions should be supported by producers that wish to handle new event types dynamically. Functions 2-6 should be supported by producers that allow consumers to initiate the flow of events, and functions 7-9 should be supported by producers that initiate the flow of events.

1. Locate Event: search the directory service for the description of an event.
2. Locate Consumer: search to the directory service for a consumer. Corresponds to Directory Service *Search*.
3. Register: add/remove/update entry(ies) in the directory service describing events that the producer will accept from the consumer. Corresponds to Directory Service *Add, Remove,* and *Update*.
4. Accept Query: accept a query request from a consumer. One or more event(s) are returned in the reply. Corresponds to Consumer *Initiate Query*.
5. Accept Subscribe: accept a subscribe request from a consumer. Further details about the event stream are returned in the reply. Corresponds to Consumer *Initiate Subscribe*.
6. Accept Unsubscribe: accept an unsubscribe request from the consumer. If this succeeds, no more events will be sent for this subscription. Corresponds to Consumer *Initiate Unsubscribe*.
7. Initiate Query: send a single set of event(s) to a consumer as part of a query "request". Corresponds to Consumer *Accept Query*.
8. Initiate Subscribe: request to send event(s) to a consumer, which are delivered in a stream. Further details about the event stream are returned in the reply. Corresponds to Consumer *Accept Subscribe*.
9. Initiate Unsubscribe: terminate a subscription to a consumer. If this succeeds, no more data will be sent for this subscription. Corresponds to Consumer *Accept Unsubscribe*.

Producers can deliver events in a stream or as a single response per request. In streaming mode a virtual connection is established between the producer and consumer and events can be delivered along this connection until an explicit action is taken to terminate it. In query mode the event is delivered as part of the reply to a consumer-initiated query, or as part of the request in a producer-initiated query.

Producers are also used to provide access control to the event data, allowing different access to different classes of users. Since Grids typically have multiple organizations controlling the resources being monitored, there may be different access policies (firewalls possibly), different frequencies of measurement, and different performance details for consumers "inside" or "outside" the organization running the resource. Some sites may allow internal access to real-time event streams, while providing only summary data off-site. The producers would enforce these types of policy decisions. This mechanism is especially important for monitoring clusters or computer farms, where there may be a large amount of internal monitoring, but only a limited amount of monitoring data accessible to the Grid.

3.2.1 Optional Producer Tasks

There are many other services that producers might provide, such as event filtering and caching. For example, producers could optionally perform any intermediate processing of the data the consumer might require. A consumer might request that a prediction model be applied to a measurement history from a particular sensor, and then be notified only if the predicted performance falls below a specified threshold. The producer might in this case filter the data for the consumer and deliver it according the schedule the consumer determines. Another example is that a consumer might request that an event be sent only if its value crosses a certain threshold. Examples of such a threshold would be if CPU utilization becomes greater than 50%, or changes by more than 20%. The producer might also be configured to compute summary data. For example, it can compute 1, 10, and 60 minute averages of CPU usage, and make this information available to consumers. Information on which services the producer provides would be published in the directory server, along with the event information.

3.3 Consumer

A consumer is any program that receives event data from a producer. The functions supported by consumers are listed below. All consumers that handle new event types dynamically should support the first function. Functions 2-5 should be supported by consumers that initiate the flow of events; functions 6-8 should be supported by consumers that allow a producer to initiate the flow of events.

1. Locate Event: search the schema repository for the event schema. The schema respository may be the GMA directory service.
2. Locate Producer: Search to the directory service for a producer. Corresponds to Directory Service *Search*.
3. Initiate Query: request event(s) from a producer, which are delivered as part of the reply. Corresponds to Producer *Accept Query*.
4. Initiate Subscribe: request event(s) from a producer, which are delivered in a stream. Further details about the event stream are returned in the reply. Corresponds to Producer *Accept Subscribe*.
5. Initiate Unsubscribe: terminate a subscription. If this succeeds, no more data will be accepted for this subscription.
6. Register: add/remove/update entry(ies) in the directory service describing events that the consumer will accept from the producer. Corresponds to Directory Service *Add*, *Remove*, and *Update*.
7. Accept Query: accept a query request from a producer. The "query" will also contain the events. Corresponds to Producer *Initiate Query*.
8. Accept Subscribe: Accept a subscribe request from a producer. Further details about the event stream are returned in the reply. Corresponds to Producer *Initiate Subscribe*.
9. Accept Unsubscribe: Accept an unsubscribe request from the producer. If this succeeds, no more events will be accepted for this subscription. Corresponds to Producer *Initiate Unsubscribe*.

There are many possible types of consumers. These include:

- Archiver: aggregate and store event data for later retrieval and/or analysis. An archiver subscribes to producers, receives event data, and places it in long -term storage. We note that many monitoring systems will need this component, as it is important to archive event data in order to provide the ability to do historical analysis of system performance, and determine when/where changes occurred. While it may not be desirable to archive all monitoring data, it is desirable to archive a good sampling of both "normal" and "abnormal" system operation, so that when problems arise it is possible to compare the current system to a previously working system. Archives may also act as a GMA producers to make the data available to other consumers.

- Real-time monitor: collect monitoring data in real time for use by real -time analysis tools. A real-time monitor searches the directory service, subscribes to all events of interest, and receives one or more streams of event data. In this way, data from many sources can be aggregated for real-time performance analysis.
- Overview monitor: This consumer collects events from several sources, and uses the combined information to make some decision that could not be made on the basis of data from only one producer. For example, one may want to trigger a call to the system administrator's pager at 2:00am only if both the primary and backup servers are down.

3.4 Compound Producer/Consumer

There may also be components that are both consumers and producers. For example a consumer might collect event data from several producers, and then use that data to generate a new derived event data type, which is then made available to other consumers, as shown in Figure 2.

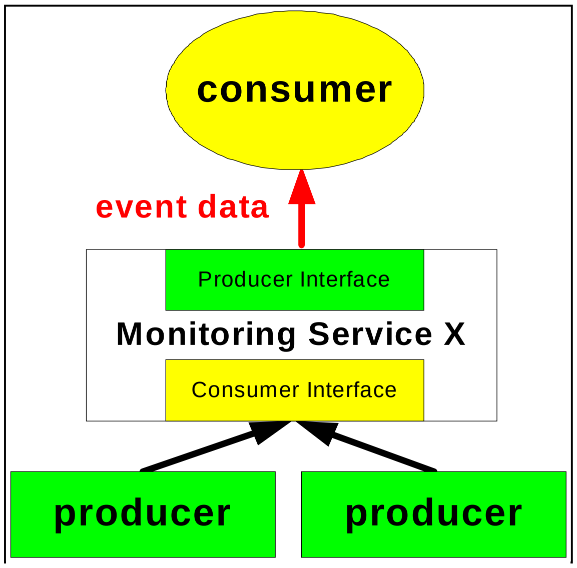

Figure 0: Compound Producer/Consumer

3.5 Sources of Event Data

There are many possible sources of the underlying data used to construct events. One possible source is some program, called a sensor, which samples a resource such as a host or network link in real-time. Another possible source is a database with a query interface, which can provide historical data. Entire monitoring systems, such as the Network Weather Service[16], could serve as a source of events. Finally, application timings from tools such as Autopilot[9] or NetLogger[13] could provide events related to a specific application. The relationship of an event source to the GMA is shown below.

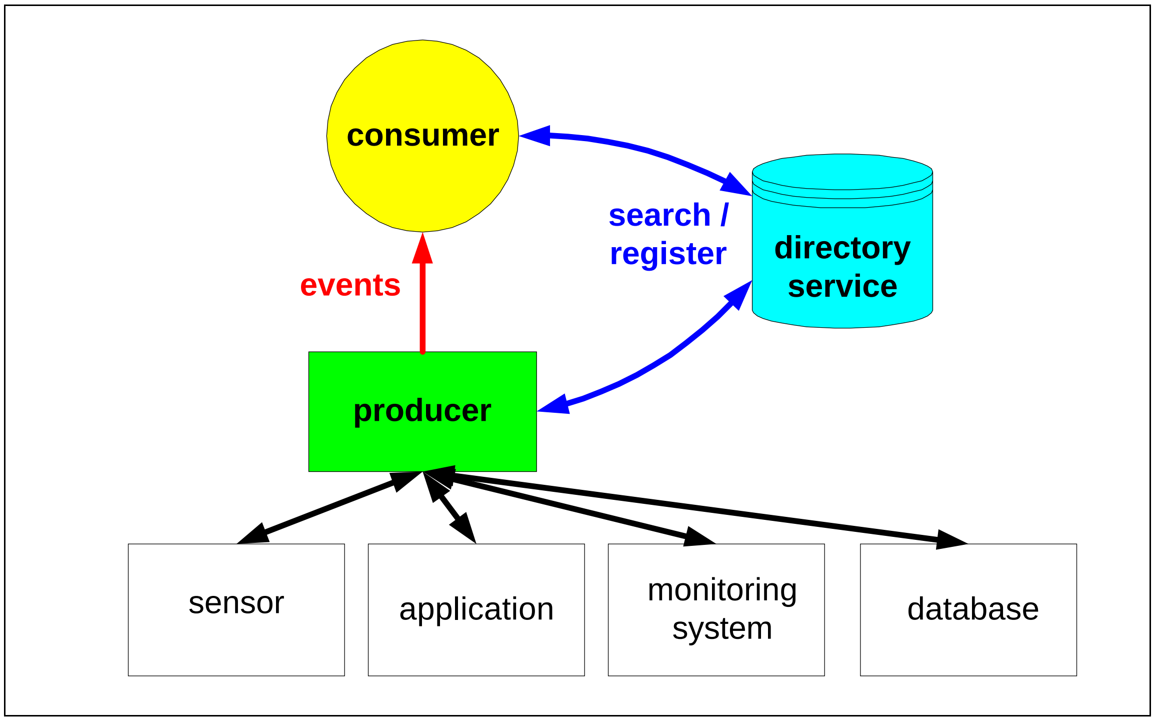

Figure 1: Sources of Event Data

A producer may be associated with a single sensor, all sensors on a given host, all sensors on a given subnet, or an arbitrary group of sensors. This is not defined by the architecture, but is left as an implementation decision. Figure shows one example of how this might be implemented. We note that there are scalability and reliability issues with how this is implemented, as described below.

4 Sample Use

An example use of the GMA is shown in Figure . Event data is collected on each host and at all network routers between them, and aggregated at a producer, which registers the availability of the data in the directory service. A real-time monitoring consumer subscribes to all this event data for real-time visualization and performance analysis. The producer is capable of computing summaries of network throughput and latency data, enabling a "network-aware" client [12] to optimally set its TCP buffer size. A subset of the producer's data, that from from the "server" and "router" nodes, is also sent to an archive.

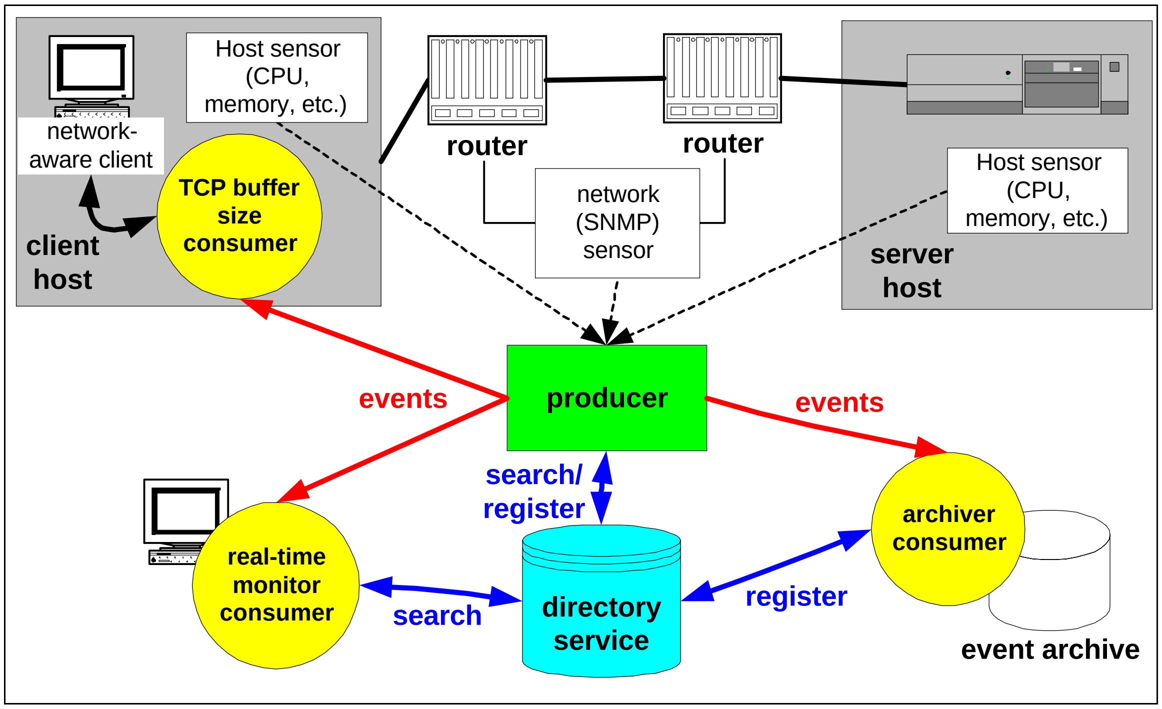

Figure 2: Sample GMA Usage

5 Implementation Issues

The purpose of a monitoring system is to reliably deliver timely and accurate information without perturbing the system. Therefore the architecture must consider performance issues explicitly and make recommendations and requirements of implementations with the goal of avoiding services which look good on paper but which fail in practice. We discuss several of these implementation design issues below.

5.1 Monitoring service characteristics

The following characteristics distinguish performance monitoring information from other system data, such as files and databases.

- Performance information has a fixed, often short lifetime of utility. Most monitoring data may go stale quickly making rapid read access important, but obviating the need for long-term storage. The notable exception to this is data that gets archived for accounting or post-mortem analysis.
- Updates are frequent. Unlike the more static forms of "metadata," dynamic performance information is typically updated more frequently than it is read. Most extant information-base technologies are optimized for query and not update, making them potentially unsuitable for dynamic information storage.
- Performance information is often stochastic. It is frequently impossible to characterize the performance of a resource or an application component using a

single value. Therefore, dynamic performance information may carry quality-of-information metrics quantifying its accuracy, distribution, lifetime, etc., which may need to be calculated from the raw data.

- Data gathering and delivery mechanisms must be high -performance. Because dynamic data may grow stale quickly, the data management system must minimize the elapsed time associated with storage and retrieval. Note that this requirement differentiates the problem of dynamic data management from the problem of providing an archival performance record. The elapsed time to read an archive, while important, is often not the driving design characteristic for the archival system. We believe that archival data is useful both for accounting purposes and for long-term trend analysis. It is our belief, however, the separate but complimentary systems for managing and archiving Grid performance data respectively are required, each tailored to meet its own set of unique performance constraints.
- Performance measurement impact must be minimized. There must be a way for monitoring facilities to be able to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.

5.2 General Implementation Strategies

A number of the authors of this white paper have built various monitoring systems. The following lessons have been learned from this experience, and should be considered when implementing a monitoring system.

The data management system must adapt to changing performance conditions dynamically. Dynamic performance data is often used to determine whether the shared Grid resources are performing well (e.g. fault diagnosis) or whether Grid load will admit a particular application (e.g. resource allocation and scheduling).
To make an assessment of dynamic performance fluctuation available, the data management system cannot, itself, be rendered inoperable or inaccessible by the very fluctuations it seeks to capture. As such, the data management system must use the data it gathers to control its own execution and resources in the face of dynamically changing conditions.

Dynamic data cannot be managed under centralized control. Having a single, centralized repository for dynamic data (however short its lifespan) causes two distinct performance problems. The first is that the centralized repository for information and/or control represents a single-point-of-failure for the entire system. If the monitoring system is to be used to detect network failure, and a network failure isolates a centralized controller from separate system components, it will be unable to fulfill its role.
All components must be able to function when temporarily disconnected or unreachable due to network or host failure. For example, a producer must still be able to accept connections from consumers even if its connection to sensors or the directory server is down. In addition, once access is restored, producers must be able to reconfigure

themselves automatically with respect to the rest of the running service components. A second problem with centralized data management is that it forms a performance bottleneck. For dynamic data, writes often outnumber reads. That is, performance data may be gathered that is never read or accessed since demand for the data cannot be predicted. Experience has shown that a centralized data repository simply cannot handle the load generated by actively monitored resources at Grid scales. All system components must be able to control their intrusiveness on the resources they monitor. Different resources experience varying amounts of sensitivity to the load introduced by monitoring. A two megabyte disk footprint may be insignificant within a 10 terabyte storage system, but extremely significant if implemented for a palm-top or RAM disk. In general, performance monitors and other system components must have tunable CPU, communication, memory, and storage requirements.

Efficient data formats are critical. In choosing a data format, there are trade offs between ease-of-use and compactness. While the easiest and most portable format may be ASCII text including both event item descriptions and event item data in each transmission, this also the least compact. This format may be suitable for cases where a small amount of data is recorded and transmitted infrequently. However, some sources of event data can generate huge volumes of data in a short amount of time, demanding that a more efficient data format be adopted. Compressed binary representations that can be read on machines with different byte orders is one possibility. Transmitting only the item data values and using a data structure obtained separately to interpret the data is another way to reduce the data volume. XML is an emerging standard that allows the data description to be separated from the data values. The XML schema could be placed in a separate directory server, retrieved, and used in conjunction with the event data values. Another possibility is to send the data descriptor one time when a consumer subscribes to a producer, and send only the data values for each event transmission. The GMA could support registration of a data format for each event, allowing different events to use the format most appropriate for their needs. Consumers could be provided plug-in modules to convert from one format to another.

5.3 Scalability

One of the biggest issues in defining a monitoring architecture for use in a Grid environment is scalability. It is critical that the act of monitoring has minimal affect on the systems being monitored. In this model, one can add additional producers and additional directory servers as needed, reducing the load where necessary. In the case where many consumers are requesting the same event data, the use of a producer reduces the amount of work on and the amount of network traffic from the host being monitored. As such, the resources that a producer will use must, themselves, be scheduled. A producer might run on a separate host from the Grid resources, to ensure that the load from the producer does not affect what was being monitored.

In particular, we believe that the GMA is more scalable than the CORBA Event Service. In the GMA, event data is not sent anywhere unless it is requested by a consumer. Many of the current event service systems, including CORBA, send all event data to a central component, which consumers then contact. In the GMA, only event data subscription information (i.e.: which producer to contact) is sent to a central directory server. Event data goes directly from producer to consumer. We believe this model will scale much better in a Grid environment.

In addition, for the GMA system to scale, performance monitoring consumers (particularly those that require the cooperation between two or more producers) must coordinate their interactions to control intrusiveness. For example, if network performance is to be monitored between all pairs of hosts attached to a single Ethernet segment, the network probes required to generate end-to-end measurements cannot occur simultaneously. If they do, both the quality of the readings that are gathered and the network capacity that is available for other work will suffer. If performance monitors are not coordinated in the Grid, the intrusiveness of performance monitoring may strongly impact available performance, particularly as the system scales. That is, if all performance facilities operate their own monitoring sensors, Grid resources will be consumed by the monitoring facilities alone. Coordinating a Grid-wide collection of sensors is complicated both by the scale of the problem (there are many Grid resource characteristics to monitor) and by the dynamically changing performance and availability of Grid resources that are being used to implement the dynamic data management service.

One recommended producer service that is important for system scalability is that of consumer-specified caching. Often a consumer needs to access only a small subset of the global data pool, and will sacrifice fast access for tight data consistency. An automatic program scheduler, for example, might want the "freshest" data that can be delivered for a specified set of hosts with no more than a one second access delay. To achieve this functionality at Grid scales, producers must cache the data the consumer will want and deliver whatever data is available at the time of request. Experience with dynamic program scheduling indicates that this type of producer is valuable to scalable performance within the Grid [5].

5.4 Security Issues

A distributed system such as this creates a number of security vulnerabilities which must be analyzed and addressed before such a system can be safely deployed on a production Grid. The users of such a system are likely to be remote from the machines being monitored and to belong to different organizations.

Typical user actions will include queries to the directory service concerning event data availability, subscriptions to producers to receive event data, and requests to instantiate new event monitors or to adjust collection parameters on existing monitors. In each case, the domain that is being monitored is likely to want to control which users may perform which actions.

Public key based X.509 identity certificates [6] are a recognized solution for cross-realm identification of users. When the certificate is presented through a secure protocol such as SSL (Secure Socket Layer), the server side can be assured that the connection is indeed to the legitimate user named in the certificate.

User (consumer) access at each of the points mentioned above (directory lookup and requests to a producer), would require an identity certificate passed though a secure protocol, e.g. SSL. A wrapper to the directory server and the producer could both call the same authorization interface with the user's identity and the name of the resource the user wants to access. This authorization interface could return a list of allowed actions, or simply deny access if the user is unauthorized. Communication between the producer and the sensors may also need to be controlled, so that a malicious user can not communicate directly with the monitoring process.

6 Related Work

There are many existing systems with an event model similar to the one described here. CORBA includes an "event service" [1] that has a rich set of features, including the ability to push or pull events, and the ability for the consumer to pass a filter to the event supplier. JINI also has a "Distributed Event Specification" [7], which is a simple specification for how an object in one Java™ virtual machine (JVM) registers interest in the occurrence an event occurring in an object in some other JVM, and then receives a notification when that event occurs. There are also several other systems with alternative event models, such as the Common Component Architecture; many of which are summarized in [8]. However, we believe that none of the existing systems is a perfect match for a Grid monitoring system; therefore we have tried to combine the relevant strengths of each. Another related system is Autopilot [9], which has had the notion of sensors for several years, and which implements a similar publish/lookup/subscribe architecture. Note that this list of systems is not intended to be exhaustive, but only illustrative of the usefulness of the proposed architecture.

7 Acknowledgements

Input from many people went into this document, including almost all attendees of the various Grid Forum meetings. The LBNL portion of this paper was supported by the U. S. Dept. of Energy, Office of Science, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division, under contract DE-AC03-76SF00098 with the University of California.

8 References

[1] CORBA, "Systems Management: Event Management Service", X/Open Document Number: P437, http://www.opengroup.org/onlinepubs/008356299/

[2] H. Dail, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw "Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem" , Proceedings of the 9th Heterogeneous Computing Workshop, May 2000.
[3] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations". In Proceedings 6th IEEE Symposium on High Performance Distributed Computing, August 1997.
[4] The Globus project: See http://www.globus.org
[5] I. Foster, C. Kesselman, eds., "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann, Pub. August 1998. ISBN 1-55860-475-8.
[6] R. Housely, W. Ford, W. Polk, D. Solo, "Internet X.509 Public Key Infrastructure", IETF RFC 2459. Jan. 1999
[7] Jini Distributed Event Specification", http://www.sun.com/jini/specs/
[8] X. Peng, "Survey on Event Service," http://www-unix.mcs.anl.gov/~peng/survey.html
[9] R. L. Ribler, J. S. Vetter, H. Simitci, and D. Reed, "Autopilot: Adaptive Control of Distributed Applications," Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.
[10] W. Smith, "Monitoring and Fault Management", http://www.nas.nasa.gov/~wwsmith/mon_fm
[11] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, M. Thompson, "A Monitoring Sensor Management System for Grid Environments", Proceedings of the IEEE High Performance Distributed Computing conference (HPDC-9), August 2000, LBNL-45260.
[12] B. Tierney, J. Lee, B. Crowley, M. Holding, J. Hylton, F. Drake, "Network-Aware Distributed Storage Cache for Data Intensive Environments", Proceeding of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896. http://www-didc.lbl.gov/DPSS/
[13] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter, "The NetLogger Methodology for High Performance Distributed Systems Performance Analysis", Proceeding of IEEE High Performance Distributed Computing conference, July 1998, LBNL-42611. http://www-didc.lbl.gov/NetLogger/
[14] W. Waheed, J. Smith, J. George, Yan. "An Infrastructure for Monitoring and Management in Computational Grids." In Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems.
[15] M. Wahl, T. Howes, S. Kille,"Lightweight Directory Access Protocol (v3)", Available from ftp://ftp.isi.edu/in-notes/rfc2251.txt
[16] R. Wolski, N. Spring, J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", Future Generation Computing Systems, 1999. http://nsw.npaci.edu/